Coding Rules for AgALag-compliant AL3101 effects

G. Soyez (Axoris)

Version: 1.2
Creation date: 20/11/01
Last modification date: 17/04/2004

Contents

History of changes

Revision number	Editor	Description of changes
1.0	Lall	Initial revision
1.1	Lall	Added remarks from Gregor's review
1.2	Gregor	Compatibility with initial SF AgALag

1 Introduction

This document describes some basic coding rules for developping AgALag-compliant modules. The final aim of these rules is to allow easy integration of codes written by different developpers and to allow building one module from different parts using AgALag. The rules are not intended to make life of the developper more complex however using these rules may add some extra complexity at the moment of writting. Anyhow, the added value of these rules when trying to build up a library of code should make them a logical step into development.

In this document, the word *module* refers to sound processing modules like an equalizer as well as to basic building functions like biquad, sine generator, etc...

Some practical aspects of use of these rules are covered at the end of the document. Some of these aspects are strongly related to the use of the forecoming AL3101 simulator.

2 Layout of a module

2.1 Header of a module

Each module should begin with some comment lines. The following information should be available from these lines:

1. The name and a short description of the module.
2. The memory and cycles requirements.
3. A descrition of the number of IOs the module is working with.
4. The name of the module developper.
5. The date and version number of the module.

The scheme for such a header is as follows:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Effect Name :
; Description :
;
; Nb Input    :
; Nb Output   :
;
; Input       : number of inputs expected by the module
; Output      : number of outputs generated by the module
;
; Cycles      : ? + <init?> cycles (-1=unknownw)
; Memory      : ? words (without INIT)
; Delay Line  : no/yes (remove wrong one)
;
; Author      :
; Date        :
; Version     :
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

2.2 Sections in a module

Each module should have the same layout and should contain different sections:

1. Section 1: Control parameters
2. Section 2: Initial values of control parameters
3. Section 3: Constants definitions
4. Section 4: Memory variables definition
5. Section 5: Initial values of memory variables
6. Section 6: Initialization
7. Section 7: Module code

Here is an in-depth description of all sections

2.2.1 Section 1: Control parameters.

Control paremeters are variables, coefficients, etc... that are supposed to be modifiables by a user. It can be the value of a scaler, coefficients of a tunable equalizer, etc... These parameters will be stored in data RAM instead of being hardcoded within the code.

The definition of these parameters should be done by means of ABS statement.

2.2.2 Section 2: Initial values of control parameters.

The initial values of the control parameters defined in section 1 should be defined here. This is done with EQU statements. The name should be the same as the one used in section 1 but preceeded by an "I". These values will be used by the initialization section.

2.2.3 Section 3: Constants definitions.

Constants values are variables, coefficients, etc... that are not supposed to be modified after assembling. This can be a fixed amplification factor, the length of a delay line, etc...

These constants will be directly used in the code.

2.2.4 Section 4: Memory variables definition.

Memory variables are internal variables to the module that are not supposed to be modified by a user. This can be filter taps, a delay line, etc...

These variables are directly used in the code.

IMPORTANT NOTE: These memory variables should be used for information that need to be kept from one cycle to another (*e.g.* previous input for an IIR filter). If you need some temporary memory for internal computation during one cycle, you should use the `DIRx` addresses.

2.2.5 Section 5: Initial values of memory variables.

In some cases, it might be useful to initialize the values of the memory variables to a known state. This can be used for example with filter taps that should be initialized to zero in order to avoid possible unstabilities.

The initial value will be given by means of EQU statements. The name should be the same as the one used in section 1 but preceeded by an "I". It must be clear that some of the values only requires initialization; it is thus not mandatory.

These values will be used by the initialization section.

2.2.6 Section 6: Initialization.

The initialization of the control parameters and of some of the memory variables is done in this section. This is done in three phases:

1. Check the INIT variable address. If set to 0, the initialization procedure should take place, otherwise skip the initialization part.
2. Initialize the control parameters with their initial values.
3. Initialize the memory variables which have an initial value defined.
4. Set the INIT address to one to avoid re-initialisation during netx cycle.

The INIT section should therefore look like this:

```
CM 0x40000 INIT
SKIP !Z EndInit
C IParam
SCA 0x0 Param
...
C 0x1000000
SCA 0x0 INIT
EndInit:
```

2.2.7 Section 7: Module code.

This section will contain the "real" processing part of the module. Remarks:

- This part should NEVER use any fix reference to the data memory (addresses 0x0 to 0x3FF). You must always refer to these addresses using the memory labels defined in sections 1 and 4.
- The input (`IN1` to `IN8`) and output (`OUT1` to `OUT8`) addresses must be referred by their name and not by their address.
- If you need temporary memory for the purposes of a computation inside one cycle, you should use the `DIRx` addresses.

- If you want to use directly the address pointer for one of the memory labels `Label`, you can use `@Label`. Note that you should use this with great caution and only use it for modules entering in an AgALag structure, this will not work for a standalone application.

3 Pratical aspects of use

3.1 Never ending typing

These rules are not supposed to make the life of the developper more complex. This is however partly the result achieved by the use of so long labels, variables, etc... Indeed, this increase dramatically the typing needed for a complete module.

As a consequence, the Axoris Development Environment AgALag will take care of helping the developper in that boring task. It is the aim that the AgALag will ask the developper his name and the name of his module so that labels, variable names, etc... will be automatically updated.

3.2 Loss of cycles in initialization

The number of cycles that will be spent in initialization may become quite huge if filter coefficients have to be initialized. In this case, it would be better to download the values directly in memory from an off-chip EEPROM or micro-controller. As a consequence, the Axoris assembler will provide an option to generate a second object file containing an image of the data memory at initialization time. The assembler will take care of removing the initialization section from the program code.

A Module template

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Effect Name :
;
; Description :
;
; Nb In       :
; Nb Out      :
;
; Input       :
; Output      :
;
; Cycles      : ? + <init?> cycles (-1=unknownw)
; Memory      : ? words (without INIT)
; Delay Line  : no/yes (remove wrong one)
;
; Author      :
; Date        :
; Version     :
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

ABS     INIT    0x3FF

        ;; SEC1 Control parameters

        ;; SEC2 Initial values of control parameters (S3.24 format)

        ;; SEC3 Constants definitions

        ;; SEC4 Memory variables definition

        ;; SEC5 Initial values of memory variables

        ;; SEC6 Initialization
CM      0x40000 INIT
SKIP    !Z      EndInit
<<insert control parameters init here >>
<<insert memory variables init here >>
<<C Value>>
<<SCA OxO Address>>
C       0x1000000           ; 1->A
```

```
SCA     0x0     INIT        ; do not reinitialize next time
EndInit:

        ;; SEC7 Module code
```